



Defining Transactions with GUPTA[®] SQLBase[®]

From the SQLBase Advanced Topic Guide

May, 2006

GUPTA[™]

Abstract	3
Introduction	3
Physical database design	3
Data integrity	3
Defining transactions	4
<i>Transaction name, number and description</i>	<i>4</i>
<i>Transaction type and complexity</i>	<i>4</i>
<i>Transaction volumes</i>	<i>5</i>
<i>Transaction performance requirements</i>	<i>5</i>
<i>Relative priority</i>	<i>6</i>
<i>Transaction SQL statements</i>	<i>6</i>
Conclusion	7



Abstract

In this paper, we will explain the benefits of well-defined transactions in order to achieve performance and integrity in the database design.

Introduction

Rigorous Transaction definition is a requirement for performance in the design of the database as well as data integrity. Thus, we will describe how to define transactions, and identifies the information, which is typically included in transaction definitions.

Physical database design


The objective of physical database design is to ensure every database transaction meets or exceeds its performance requirements. Typically, database performance is measured in terms of transaction performance. The process of physical database design involves mapping the logical data structures (entities and relationships in an ER diagram) to the physical constructs of SQLBase. For example, entities in the logical database model become one or more clustered or non-clustered SQLBase tables. The design criteria guiding this mapping are transaction performance requirements and volume statistics. The deciding factor for the designer choosing between the various physical constructs that may be used is their effect on transaction performance. For any particular construct, the effect will probably enhance the performance of some transactions while degrading the performance of others. Therefore, physical database design can be viewed as an optimization problem. The goal is to select the SQLBase physical constructs so that every transaction meets or exceeds its performance requirements. Hence, well-defined transactions are a prerequisite for physical database design. Without this information, the database designer has no rule to guide in the selection of physical database constructs. For example, the database designer will be unable to determine whether a clustered or non-clustered table will result in better performance. A similar argument can be made for index selection.

Data integrity

Database transactions usually involve *multiple* database accesses. For example, consider the banking transaction of withdrawing \$1000 from a savings account and depositing the funds into a checking account. This transaction entails two separate database operations: subtracting \$1000 from the savings table and adding \$1000 to the checking table. If either one of these database operations successfully completes but the other does not, then the account balances would be wrong. Both of these operations must complete successfully or neither of them should complete successfully. Thus, a database transaction is a logical unit of work that causes the database to advance from one consistent state to another. In other words, a transaction moves the database from one state that does not violate the data integrity requirements of the

Well-defined transactions are a prerequisite for physical database design

Rigorous transaction definition for strong database integrity



database to another state that does not violate the data integrity requirements of the database. In addition, a database transaction is a grouping of logically related database operations so that all of the database accesses should complete successfully or none of them should be applied to the database. Otherwise, if only some of the database accesses complete successfully, the database is left in an inconsistent state. Therefore, in order to ensure the integrity of the database, it is necessary to rigorously define the database transactions.

Key ingredients for well defined transactions:

Defining transactions

Transaction definitions can take many forms. Some organizations may possess a CASE tool with a transaction definition facility as part of a data repository. Other organizations may choose to define transactions using paper-based or automated forms. Regardless of the media, all good transaction definitions include several key ingredients:

Transaction name, number and description

The first step in defining transactions is to uniquely identify each database transaction. This is accomplished by assigning a name and unique identifier to each transaction. In addition, a short narrative should be composed, referred to as the transaction description, which describes the transaction in business terms:

- The description should specify what the transaction accomplishes for the user rather than how it is accomplished.
- The description should be understandable by users (it should not include technical jargon).

The purpose of transaction names and descriptions is to provide users and MIS staff with a means for distinguishing one transaction from another.

Example: Transaction Number: 001

Example: Transaction Name: Transfer Funds

Example: Transaction Description: The Transfer Funds transaction verifies that sufficient funds exist in the account to be debited. If sufficient funds exist, the transaction debits the account by the specified amount and credits the other account for the same amount.

Uniquely identify each database transaction

In Online Transaction Processing (OLTP) systems, the database transactions are known ahead of time. For these types of systems, a one-to-one relationship will exist between the transaction definitions and the actual transactions submitted to SQLBase. In Decision Support Systems (DSS), transactions are not known ahead of time. These systems are characterized by ad hoc and exception reporting. Therefore, it is not possible to describe each and every transaction. With these systems, the transaction definitions are illustrative of the actual transactions, which will be submitted to SQLBase once the system is implemented. DSS systems require the database designer to predict the type of transactions which the users will likely run.

Transaction type and complexity

Frequently, transaction definitions also categorize transactions in terms of type and complexity. For example, transactions may be defined as



Identify transaction complexity

either "Batch" or "Online" and given a complexity rating of either "High," "Medium," or "Low." This information is useful for producing listings or totals of like transactions. For example, management may require an estimate of the time required to complete physical database design. To accomplish this, one might need to know how many low, medium, and high complex transactions a system has. The database for a system with a large percentage of high complexity transactions would take longer to design than a system with mostly low complexity transactions. The complexity rating is a matter of judgment, but should be based on the degree of difficulty the database designer is likely to have meeting the performance requirements for the transaction. A transaction of high complexity has at least two of the following characteristics:

- Contains many SQL statements (more than 10).
- Contains WHERE clauses with many predicates (for example, three or more table joins or sub queries).
- The SQL statements affect many rows (> 100).

On the other hand, a low complexity transaction has the following characteristics:

- Contains few SQL statements (three or fewer).
- Contains WHERE clauses with only one or two predicates.
- The SQL statements affect few rows (< 25).

Document Transaction volume


Transaction volumes

Transaction definitions must also include volume information. This typically includes the average and peak frequency of each transaction. For example, a branch bank might estimate that the funds transfer transaction described in the introduction of this chapter will occur 50 times an hour with peak loads reaching 65 transactions per hour. The transaction volume statistics are extremely important for physical database design; consequently, care should be taken to be as accurate as possible. For example, the database designer will handle a transaction which occurs 1000 times per hour quite differently from one that will occur only once per hour. If the new computer system is replacing an existing system, it may be possible to derive transaction volumes from the existing system. On the other hand, new systems rarely duplicate the transaction logic of their predecessors so these differences should be considered. Typically, the hardest situation in which to deduce accurate transaction volumes is one in which no automated or manual predecessors exist. These situations occur when organizations enter new lines of business. In this case, volume statistics may be no more than educated guesses

Verify transaction performance requirements

Transaction performance requirements

The transaction definitions should also document the performance requirements of each transaction. As mentioned in the introduction, the transaction performance requirements are the basis of physical database design. During physical database design, the database designer chooses the physical constructs from among the various options provided by SQLBase, such that all transactions meet or exceed their transaction performance requirements. Without specifying performance requirements, the database designer has no means by



which to evaluate whether or not the database performance is acceptable. Some organizations prefer to specify exact performance requirements for each transaction. The performance requirements typically take the form of number of execution seconds. For example, one requirement might state that the transfer funds transaction must complete in three seconds or less. Other organizations find it easier to define performance requirements in terms of transaction types and complexity. For example, an organization might require:

- high complexity, online transactions to execute in ten seconds or less.
- medium complexity, online transactions to execute in six seconds or less.
- low complexity, online transactions to execute in three seconds or less.
- high complexity, batch transactions to execute in 60 minutes or less.
- medium complexity, batch transactions to execute in 30 minutes or less.
- low complexity, batch transactions to execute in 15 minutes or less.

Relative priority


Transaction definitions also should rank each transaction in terms of relative priority, which describes how important to the business one transaction is compared to all other transactions. Physical database design ensures that each transaction meets or exceeds its performance requirements. Consequently, the transactions must be evaluated one at a time. The relative priority of each transaction provides the sequence in which transactions are evaluated. This ensures that the most important transactions receive the most attention. In addition, during the process of physical database design, the database designer will tailor the physical database design so that one given transaction executes faster. Making one transaction go faster frequently negatively affects the performance of other transactions. Relative priority ranking also provides the basis for any transaction arbitration scheme. The ranking can take many forms. For example, some organizations assign a number from one (high) to ten (low) to each transaction.

Transaction SQL statements

Each transaction definition must also specify the SQL statements that perform the necessary database operations. These SQL statements are UPDATE, INSERT, DELETE or SELECT commands, which are known collectively as SQL DML (data manipulation language). The SQL statements are required during physical database design to forecast the execution times of the transactions. In addition, a detailed understanding of the database design and significant experience with SQL is required to accurately code many SQL statements. Therefore, it is sometimes preferable for the database designer to provide the SQL statements for the application programmers rather than allowing them to code the SQL themselves. However, the database designer and the application programmers must coordinate their efforts closely to ensure that the SQL statements support the program's data requirements. The transaction definitions should also include, for each SQL statement, a brief (non-technical) narrative explaining:

Rank transaction priority

Define, document and validate transactions SQL statement

- 
- What the command accomplishes
 - Why it is required (if it is not obvious)
 - Number of rows in the database affected by the command

The number of rows affected means either the number of rows returned (in the case of a SELECT command), or the number of rows inserted, updated, or deleted. The performance of any SQL command is dependent on the number of database rows affected. For example, it would take longer to read 10,000 customer rows than one customer row. Hence, the number of records affected by each SQL command is an important element of physical database design. Once coded and described, each SQL command should be validated for accuracy. To accomplish this, it will be necessary to construct a small test database. Many users find it useful to build a SQLTalk script containing all the transactions' SQL commands for this purpose.

Conclusion

By defining rigorous transaction definition in SQLBase that is, uniquely identify each database transaction, their complexity, documenting their volume, their individual performance requirement and ranking their priority, you will meet or exceed performance requirement in your database design as well as achieve strong data integrity.

Copyright © 2005 Gupta Technologies LLC. Gupta, the Gupta logo, and all Gupta products are licensed or registered trademarks of Gupta Technologies, LLC., All other products are trademarks or registered trademarks of their respective owners. All rights reserved.